6.6.2.2 Basic Operating System Entry Cost

System	OS entry (lat_syscall)
Linux/Alpha	2
Linux/i586	2
Linux/i686	3
Unixware/i686	4
Sun Ultra 1	5
FreeBSD/i586	6
Solaris/i686	7
DEC Alpha@300	9
Sun SC1000	9
HP K210	10
SGI Indigo 2	11
DEC Alpha@150	11
IBM PowerPC	12
IBM Power 2	16
SGI Challenge	24
DAISY systems	
FreeBSD/i586 (p5-90)	8
HEAT systems	
DEC Alpha	13
HP 9000/735	12
IBM RS6000	65
SUN SS10	36
SGI IRIX	39

Table 12. lat_syscall results (microseconds).

Entering into the operating system is required for many system facilities. The cost of entering and exiting the operating system without pause is useful when calculating the cost of a system facility Through lat_syscall, lmbench measures this basic operating system entry cost (Table 12). This is accomplished by repeatedly writing one byte to /dev/null, a pseudo device driver that does nothing but discard the data.

6.6.2.3 Process Creation Times

The lmbench process benchmark is used to measure the basic process primitives: creating a new process (fork & exit), running a different program (fork, exec, & exit), and context switching (fork, exec sh -c, & exit). The benchmarks measure the time it takes to create a basic thread of control.

fork & exit	:	The time it takes to split a process into nearly two identical copies and have one exit.
fork, exec, & exit	:	The time it takes to create a new process and have that new process run a new program.
fork, exec sh-c, & exit	:	The time it takes to create a new process and have that new process run a new program by asking the system shell to find that program and run it.

Process creation benchmarks are of particular interest to distributed systems since many remote operations include the creation of a remote process to help the remote operation to complete. Table 13 show the results of the lat_proc benchmark.

System	process creation		
	fork & exit	fork, exec, & exit	fork, exec sh-c, & exit
Linux/Alpha	0.7	3	12
linux/i686	0.4	5	14
Linux/i586	0.9	5	16
Unixware/i686	0.9	5	10
DEC Alpha@300	2	6	16
IBM PowerPC	2.9	8	50
SGI Indigo 2	3.1	8	19
IBM Power 2	1.2	8	16
FreeBSD/i586	2	11	19
HP K210	3.1	11	20
DEC Alpha@150	4.6	13	39
SGI Challenge	4	14	24
Sun Ultra 1	3.7	20	37
Solaris/i686	4.5	22	46
SUN SC1000	14	69	281
DAISY systems			
FreeBSD/i586 (p5-90)	4.7	19.9	35.1
HEAT systems			
DEC Alpha	2.7	9.9	23.5
HP 9000/735	3.4	10	19.9
IBM RS6000	3.9	12.2	28.9
SUN SS10	6.3	23.9	37.7
SGI IRIX	4.5	13.8	31.2

Table 13 show the results of the lat_proc benchmark (microseconds).

6.6.2.4 Context Switching

In a multiprocessing environment, timesharing (sharing the CPU and memory with several processes at the same time) must take place. Therefore, at any instance it must be possible to switch from one process to the next. The context switching benchmark measures the time it takes to save the state of one process and restore the state of another.

The context switch benchmark (lat_ctx) is implemented as a ring of two to twenty processes that are connected with UNIX pipes. A token is passed from process to process, forcing context switches. The benchmark measures the time it takes to pass the token two thousand times from process to process (Table 14). Each hand off of the token has two costs: (a) the context switch, and (b) the cost of passing the token. In order to get the context switching time, the benchmark first measures the cost of passing the token through a ring of pipes in a single process. This time (overhead time) is defined as the cost of passing the token and is not included in the reported context switch.

Results from a DAISy system are shown in Figure 17. As the size and number of processes increase, processes start falling out of cache, resulting in higher context switch times. Note the increase in overhead as the size increases.

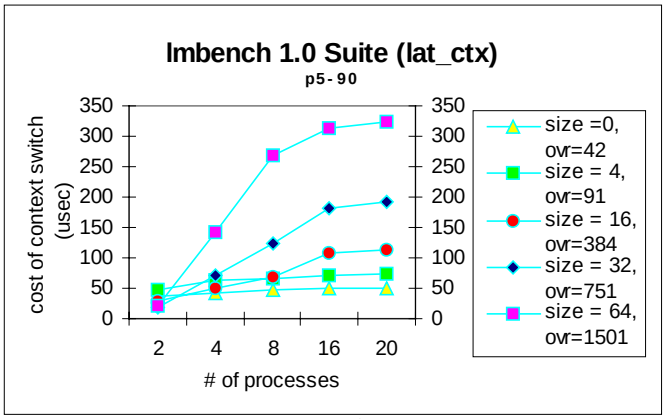

Figure 17. lat_ctx results for a DAISy system.

	context switch			
	2 processes		8 processes	
System	0KB	32KB	0KB	32KB
Linux/i686	6	18	7	101
Linux/i586	10	163	13	215
Linux/Alpha	11	70	13	78
IBM Power2	13	16	18	43
Sun Ultra1	14	31	20	102
DEC Alpha@300	14	17	22	41
IBM PowerPC	16	87	26	144
HP K210	17	17	18	99
Unixware/i686	17	17	18	72
FreeBSD/i586	27	34	33	102
Solaris/i686	36	54	43	118
SGI Indigo2	40	47	38	104
DEC Alpha@150	53	68	59	134
SGI Challenge	63	80	69	93
SUN SC1000	107	142	104	197
DAISY systems				
FreeBSD/i586 (p5-90)	37	19	47	124
HEAT systems				
DEC Alpha	45	35	55	23
HP 9000/735	20	37	24	185
IBM RS6000	106		109	
SUN SS10	34	60	79	106
SGI IRIX	90	98	94	185

Table 14. lat_ctx results (microseconds).

6.6.2.5 Interprocess Communication

The cost of communicating between processes or IPC overhead consists of the time required to execute a system call and the time to move the data between processes.

The IPC latency benchmarks (lat_connect, lat_rpc/udp, lat_rpc/tcp, lat_tcp, and lat_udp) are implemented by passing a small message (a byte or so) back and forth. The reported results are the microseconds it takes to do one round trip. Results are shown in Table 15.

System	Network	pipe	TCP		RPC/TCP		UDP		RPC/UDP		connect	
		local	local	remote	local	remote	local	remote	local	remote	local	remote
		host	host	host	host	host	host	host	host	host	host	host
Linux/i686		26	216	na	346	na	93	na	180	na	263	na
Linux/i586		33	467	na	713	na	187	na	366	na	606	na
Linux Alpha		34	429	na	602	na	180	na	317	na	na	na
Sun Ultra1	100baseT	62	162	280	346	na	197	308	267	na	852	na
IBM PowerPC		65	299	na	698	na	206	na	536	na	na	na
Unixware/i686		70	na	na	na	na	na	na	na	na	na	na
DEC Alpha@300		71	267	na	371	na	259	na	358	na	na	na
HP K210		78	146	na	606	na	152	na	543	na	238	na
IBM Power2		91	332	na	649	na	254	na	531	na	339	na
Solaris/i686		101	305	na	528	na	348	na	454	na	1230	na
FreeBSD/i586	100baseT	104	256	365	440	na	212	304	375	na	418	na
SGI Indigo2	10baseT	131	278	543	641	na	313	602	671	na	667	na
DEC Alpha@150		179	467	na	788	na	489	na	834	na	na	na
SGI Challenge		251	546	na	900	na	678	na	893	na	716	na
Sun SC1000		278	855	na	1386	na	739	na	1101	na	3047	na
DAISY systems												
FreeBSD/i586 (p5-90)	10baseT	153	407	731	740	na	340	615	615	887	609	544
FreeBSD/i586 (p5-90)	100baseT	na	442	572	na	na	378	470	664	723	650	154
HEAT systems												
DEC Alpha	fddi	146	386	567	647	na	412	1089	707	1181	3904	976
HP 9000/735	fddi	159	222	419	799	na	225	403	774	572	360	535
IBM RS6000	fddi	408	1178	2033	2185	na	936	1893	1841	2576	1076	2326
SUN SS10	fddi	175	495	1243	1036	na	515	1293	956	1662	672	1405
SGI IRIX	fddi	314	643	28716	1216	na	653	28702	1302	10972	1228	25212

Table 15. IPC latency results (microseconds).

Pipe Latency: Pipe latency is measured by creating a pair of pipes, forking a child process, and passing a word back and forth. Table 15 shows the results.

TCP and RPC/TCP: TCP and RPC/TCP connections are typically used in low bandwidth latency sensitive applications. TCP latency is measured by having a server process which waits for connections and a client process that connects to the server. The benchmark passes a token back and forth between the two processes through a TCP socket and measures the round trip time. Note that the RPC layer frequently adds hundreds of microseconds of additional latency.

UDP and RPC/UDP: UDP sockets are an alternative to TCP sockets. UDP and RPC/UDP messages are commonly used in client server applications. NFS is probably the most widely used RPC/UDP application in the world.

UDP latency is measured by having a server process which waits for connections and a client process that connects to the server. The benchmark passes a token back and forth between two processes through a UDP socket and measures the round trip time. Again, note that the RPC layer can add hundreds of microseconds of additional latency.